DW_tap_uc

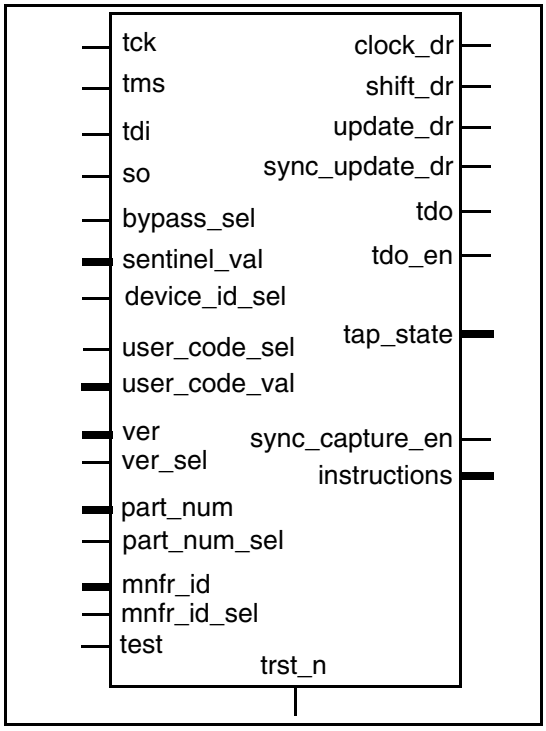
TAP Controller with USERCODE Support

Version, STAR, and myDesignWare Subscriptions: IP Directory

Features and Benefits

Revision History

- IEEE Standard 1149.1 compliant
- Synchronous or asynchronous registers with respect to `tck`
- Provides interface to supports the standard IEEE 1149.1 and optional instructions
- Optional use of device identification register and IDCODE instruction and support of USERCODE instruction
- User defined opcode for IDCODE
- Parameterized instruction register width
- External interface to program device identification register



Description

DW_tap_uc provides access to on-chip boundary scan logic and supports USERCODE instructions when device identification register is present.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
tck	1 bit	Input	Test clock
trst_n	1 bit	Input	Test reset, active low
tms	1 bit	Input	Test mode select
tdi	1 bit	Input	Test data in
so	1 bit	Input	Serial data from boundary scan register and data registers
bypass_sel	1 bit	Input	Selects the bypass register, active high
sentinel_val	*width* – 1 bits	Input	User-defined status bits
device_id_sel	1 bit	Input	Selects the device identification register, active high

Table 1-1 Pin Description (Continued)

Pin Name	Width	Direction	Function
user_code_sel	1 bit	Input	Selects the `user_code_val` bus for input in to the device identification register, active high
user_code_val	32 bits	Input	32-bit user defined code
ver	4 bits	Input	4 bit version number
ver_sel	1 bit	Input	Selects version from the parameter or the `ver` input port ■ 0 = *version* (parameter) ■ 1 = `ver` (input port)
part_num	16 bits	Input	16 bit part number
part_num_sel	1 bit	Input	Selects part from the parameter or the part_num from the input port ■ 0 = *part* (parameter) ■ 1 = `part_num` (input port)
mnfr_id	11 bits	Input	11-bit JEDEC manufacturer's identity code (mnfr_id ≠ 127)
mnfr_id_sel	1 bit	Input	Selects *man_num* from the parameter or `mnfr_id` from the input port ■ 0 = *man_num* (parameter) ■ 1 = `mnfr_id` (input port)
test	1 bit	Input	For scannable designs, the `test` pin is held active (high) during testing. For normal operation, it is held inactive (low).
clock_dr	1 bit	Output	Clocks in data in asynchronous mode
shift_dr	1 bit	Output	Enables shifting of data in both synchronous and asynchronous mode, active high
update_dr	1 bit	Output	Enables updating data in asynchronous mode, active high
tdo	1 bit	Output	Test data out
tdo_en	1 bit	Output	Enable for tdo output buffer, active high
tap_state	16 bits	Output	Current state of the TAP finite state machine
instructions	*width* bits	Output	Instruction register output
sync_capture_en	1 bit	Output	Enable for synchronous capture, active low
sync_update_dr	1 bit	Output	Enables updating new data in *sync_mode*, active high

Table 1-2 Parameter Description

Parameter	Values	Description
width	2 to 32 Default: None	Width of instruction register
id	0 or 1 Default: 0	Determines whether the device identification register is present ■ 0 = Not present ■ 1 = Present
idcode_opcode	1 to $2^{width-1}$ Default: 1	Opcode for IDCODE
version	0 to 15 Default: 0	4-bit version number
part	0 to 65535 Default: 0	16-bit part number
man_num	0 to 2047, man_num ≠ 127 Default: 0	11-bit JEDEC manufacturer identity code
sync_mode	0 or 1 Default: 0	Determines whether the bypass, device identification, and instruction registers are synchronous with respect to tck ■ 0 = Asynchronous, ■ 1 = Synchronous
tst_mode	0 or 1 Default: 1	Controls whether the `test` input is used ■ 1: The `test` input is used; (backward compatible with older versions) ■ 0: The `test` input is not used, which ensures that unused clock gating logic will not be included

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
str	Synthesis model	DesignWare or Test-IEEE-STD-1149-1

Table 1-4 Simulation Models

Model	Function
DW04.DW_tap_uc_CFG_SIM	Design unit name for VHDL simulation
dw/dw04/src/DW_tap_uc_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_tap_uc.v	Verilog simulation model source code

Control of DW_tap_uc is through the pins `tck`, `tms`, `tdi`, `tdo`, and `trst_n`. `tck`, `tms`, and `trst_n` control the states of the boundary scan test logic. `tdi` and `tdo` provide serial access to the instruction and data registers.

DW_tap_uc contains the IEEE standard 1149.1 TAP finite state machine, instruction register, bypass register, and the optional device identification register. Also, through the pins `device_id_sel`, `usr_code_sel` and `user_code_val`, the user defined USERCODE instruction is supported. For details, see Table 1-5.

Table 1-5 Device identification Register Operation

device_id_sel	user_code_sel	Selection
1	0	Selects IDCODE instruction
1	1	Selects USERCODE instruction
0	X	No operation

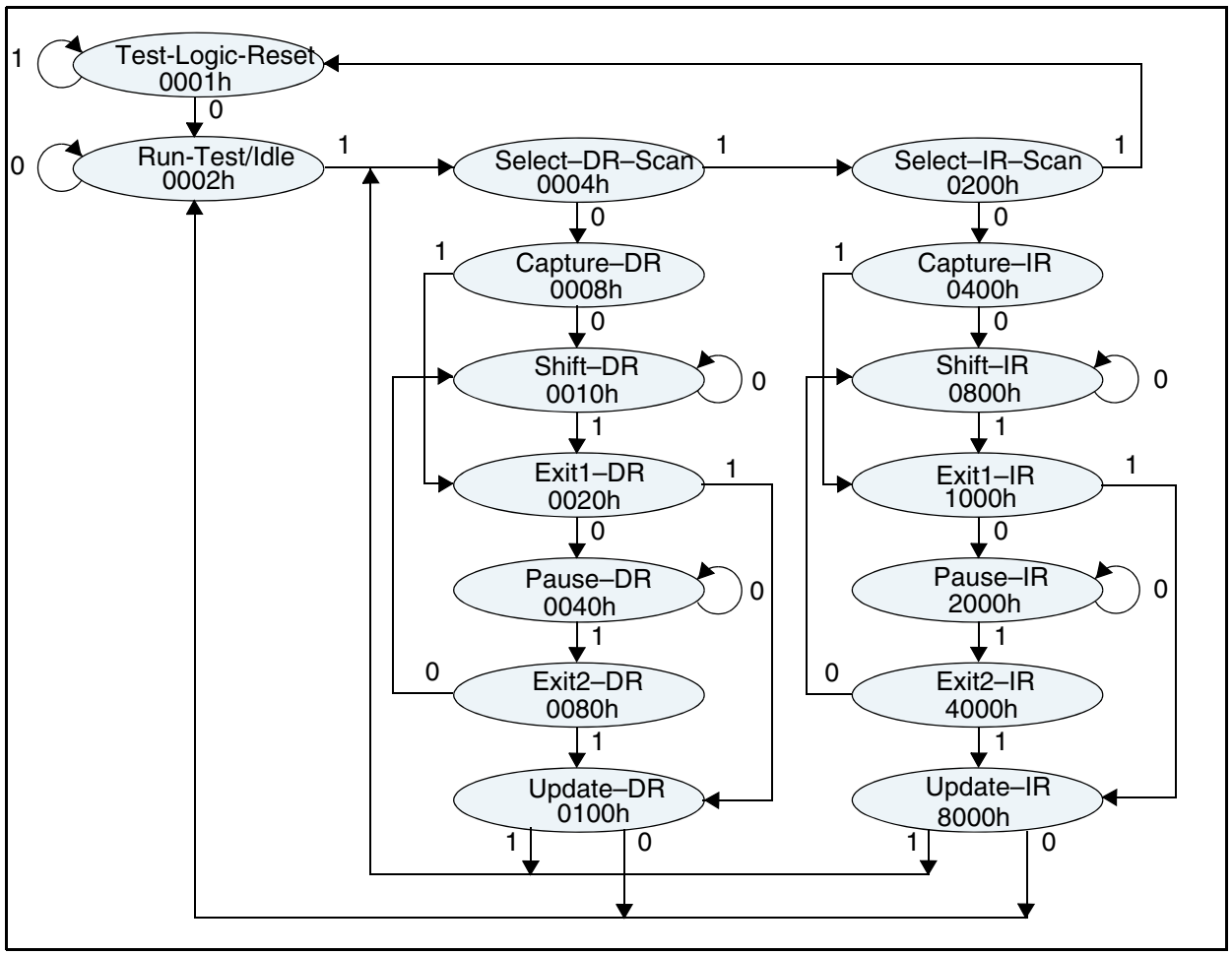
An interface port, consisting of `ver`, `ver_sel`, `part_num`, `part_num_sel`, `mnfr_id` and `mnfr_id_sel`, is provided to externally program the version, part number or the manufacturer's identity by connecting them to logic '1' or logic '0' using metal contacts. It allows the user to choose values other than the default values contained in the generic parameters that are used during initial compile.

Attention If these ports are not used, they should be connected to logic '0' so that the related logic is optimized away and the default values as defined in the generic parameters are used.

The `tck` signal is the clock for the TAP finite state machine. The data on the `tms` and `tdi` signals is loaded on the rising edge of `tck`. Data is available at `tdo` on the falling edge of `tck`. If the parameter *sync_mode* is set to 1, then `tck` is used to clock data into the instruction register, bypass register, and identification register. If *sync_mode* is set to 0, then the signals generated by the TAP finite state machine control the clocking of the registers.

The `tms` signal controls the transitions of the TAP state machine, as illustrated in the state diagram in Figure 1-1 on page 5. The state transitions occur at the rising edge of `tck`. The `tdi` signal is the serial test data input, and the `tdo` signal is the serial test data output.

Figure 1-1 State Diagram



The `trst_n` signal is an active low signal that asynchronously resets the TAP state machine to the Test-Logic-Reset state.

The DW_tap_uc output signals `clock_dr`, `shift_dr`, `update_dr`, `sync_update_dr`, and `sync_capture_en` control the boundary scan cells. In synchronous mode, boundary scan cells are updated on the rising edge of `tck`. In asynchronous mode, the boundary scan cells are updated by the rising edge of the `clock_dr` signal, asynchronous to `tck`.

The `tap_state` port provides access to the one-hot encoded TAP state machine, enabling advanced users to construct add-on circuits.

DW_tap_uc Registers

The bypass register is a mandatory IEEE 1149.1 standard 1-bit register that provides a minimum length path through the IC. The instruction value of all ones selects the bypass register as the serial connection between `tdi` and `tdo`. Other decoded instructions can select the bypass register through the `bypass_sel` input signal.

The device identification register is an optional IEEE 1149.1 standard register. If the parameter `id` is set to 1, the Design Compiler (DC) builds a 32-bit device identification register. The parameters *part*, *version*, and *man_num* determine the 32-bit number that is loaded into the device identification register when the instruction

```
000000.....01 (IDCODE instruction)
```

is selected. These values can also be loaded in to the device identification register by executing a user-defined instruction that decodes to select the `device_id_sel` port. The component also allows the manufacturer, part number, and variant for the component to be read in a serial binary form.

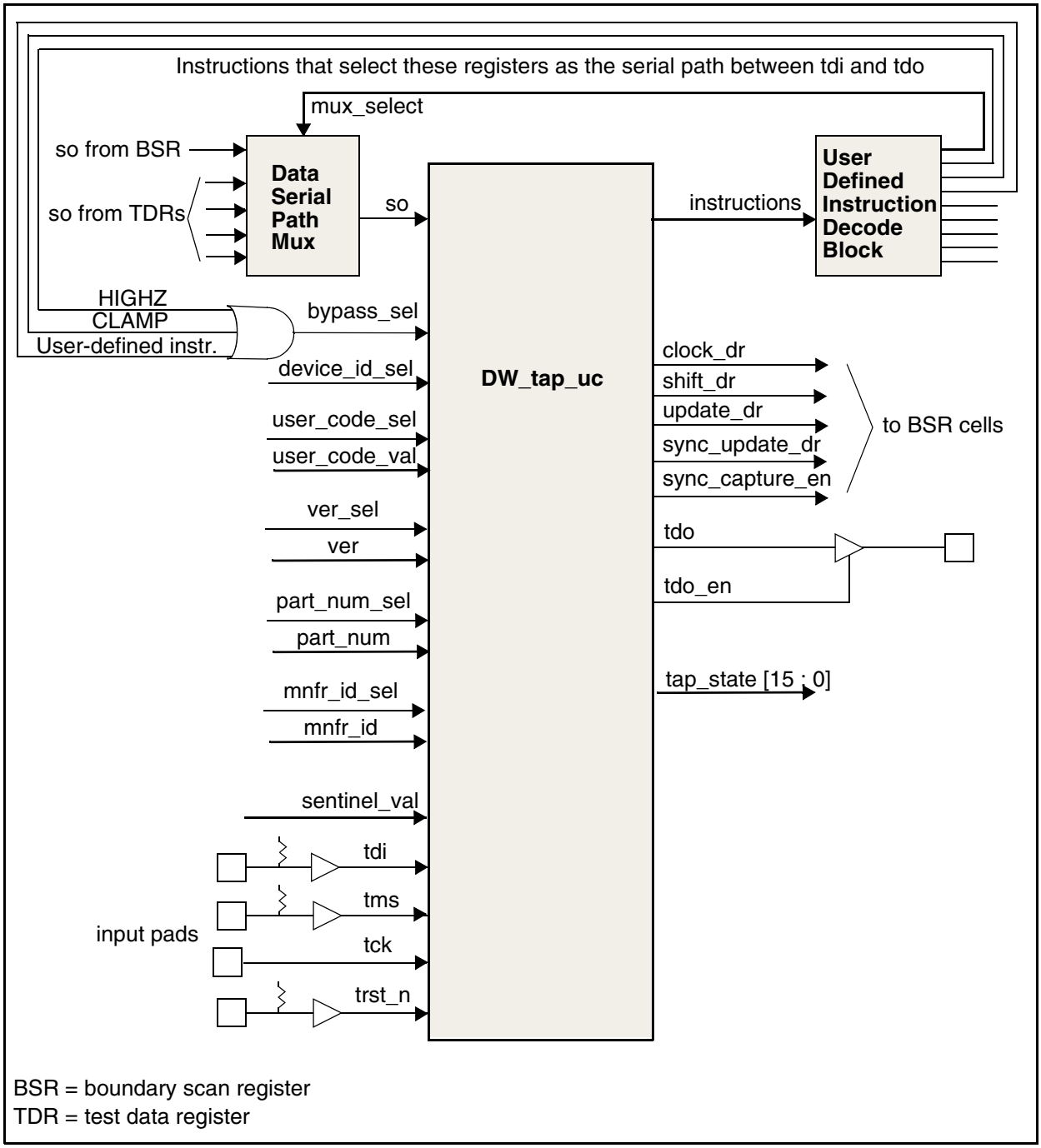
The instruction register is used to serially shift in the instructions that operate the boundary scan circuitry. The instruction register must be at least two bits wide to hold the instruction codes for the three IEEE 1149.1 standard required boundary scan instructions: BYPASS and other standard instructions. The parameter `width` determines the size of the instruction register. The output port `instructions` is provided for the user to define the format these and other user defined instructions and used appropriately in conjunction with other components of this family. The DW_tap_uc decodes only the mandatory instruction - BYPASS.

The `sentinel_val` bus provides extra status bits in the IC design. You can connect the `sentinel_val` bus to any signal you want to view when accessing the instruction register. If you do not want to use this feature, you must tie the `sentinel_val` bus to logic zero. The most significant bit of `sentinel_val` is always tied to zero.

DW_tap_uc supports user-defined test data registers. The test data registers are connected serially between `tdi` and `tdo` by a user-defined multiplexer. The multiplexer must select either the boundary scan register serial path or any of the user-defined test data registers during the appropriate instruction. The output of the multiplexer is connected to the ***so*** input of DW_tap_uc.

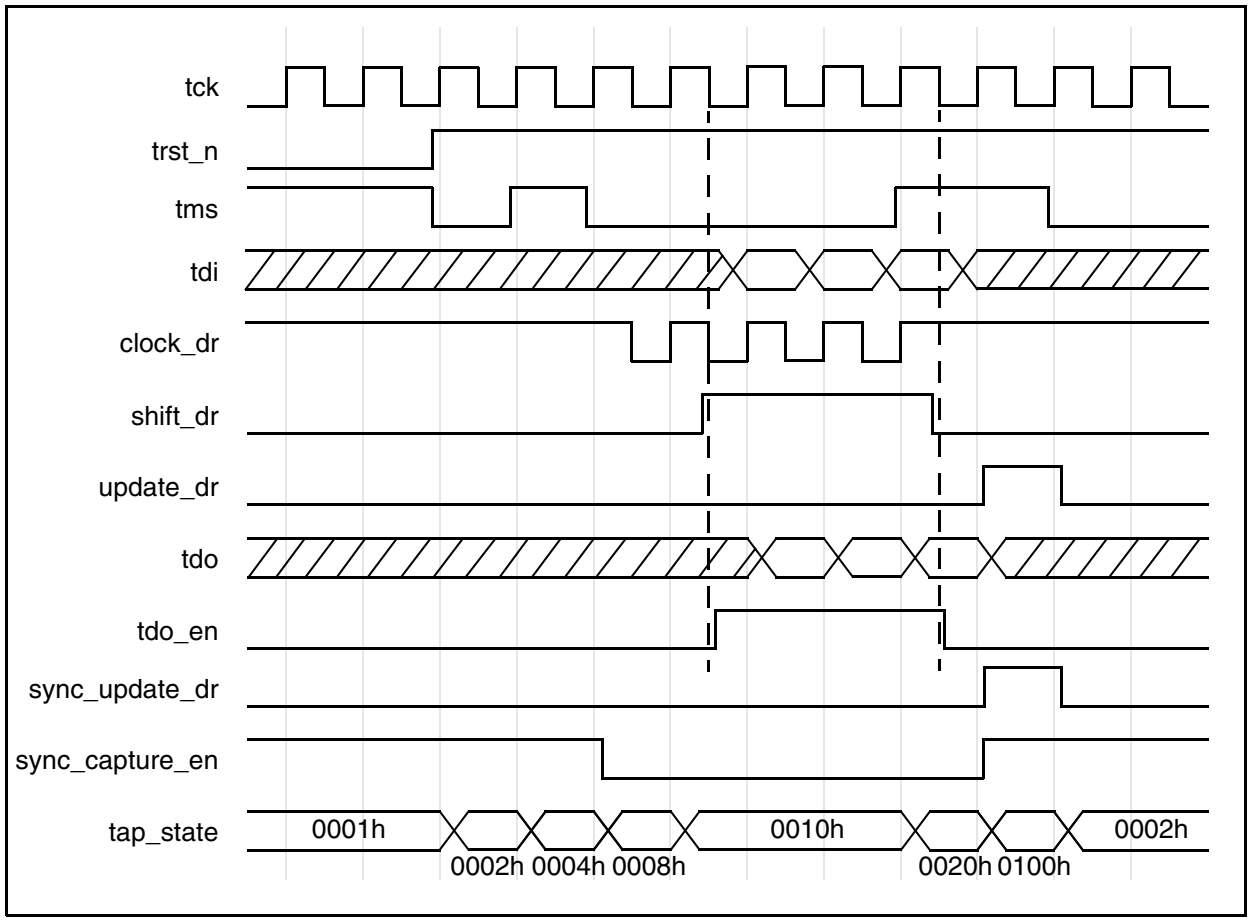
Figure 1-2 on page 7 shows the user-defined interface logic needed to implement DW_tap_uc in an integrated circuit.

Figure 1-2 Interface Diagram



Timing Diagram

Figure 1-3 Timing Diagram



Related Topics

- Application Specific – JTAG Overview
- DesignWare Building Block IP User Guide

HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DWpackages.all;
use DWARE.DW_foundation_comp.all;

entity DW_tap_uc_inst is
      generic (
        inst_width : INTEGER := 8;
        inst_id : INTEGER := 0;
        inst_idcode_opcode : INTEGER := 1;
        inst_version : INTEGER := 0;
        inst_part : INTEGER := 0;
        inst_man_num : INTEGER := 0;
        inst_sync_mode : INTEGER := 0;
        inst_tst_mode : INTEGER := 1
        );
      port (
        inst_tck : in std_logic;
        inst_trst_n : in std_logic;
        inst_tms : in std_logic;
        inst_tdi : in std_logic;
        inst_so : in std_logic;
        inst_bypass_sel : in std_logic;
        inst_sentinel_val : in std_logic_vector(inst_width-2 downto 0);
        inst_device_id_sel : in std_logic;
        inst_user_code_sel : in std_logic;
        inst_user_code_val : in std_logic_vector(31 downto 0);
        inst_ver : in std_logic_vector(3 downto 0);
        inst_ver_sel : in std_logic;
        inst_part_num : in std_logic_vector(15 downto 0);
        inst_part_num_sel : in std_logic;
        inst_mnfr_id : in std_logic_vector(10 downto 0);
        inst_mnfr_id_sel : in std_logic;
        clock_dr_inst : out std_logic;
        shift_dr_inst : out std_logic;
        update_dr_inst : out std_logic;
        tdo_inst : out std_logic;
        tdo_en_inst : out std_logic;
        tap_state_inst : out std_logic_vector(15 downto 0);
        instructions_inst : out std_logic_vector(inst_width-1 downto 0);
        sync_capture_en_inst : out std_logic;
        sync_update_dr_inst : out std_logic;
        inst_test : in std_logic
        );
    end DW_tap_uc_inst;
```

```
architecture inst of DW_tap_uc_inst is

begin

    -- Instance of DW_tap_uc
    U1 : DW_tap_uc
      generic map ( width => inst_width,
                    id => inst_id,
                    idcode_opcode => inst_idcode_opcode,
                    version => inst_version,
                    part => inst_part,
                    man_num => inst_man_num,
                    sync_mode => inst_sync_mode,
                    tst_mode => inst_tst_mode )
      port map ( tck => inst_tck,
                 trst_n => inst_trst_n,
                 tms => inst_tms,
                 tdi => inst_tdi,
                 so => inst_so,
                 bypass_sel => inst_bypass_sel,
                 sentinel_val => inst_sentinel_val,
                 device_id_sel => inst_device_id_sel,
                 user_code_sel => inst_user_code_sel,
                 user_code_val => inst_user_code_val,
                 ver => inst_ver,
                 ver_sel => inst_ver_sel,
                 part_num => inst_part_num,
                 part_num_sel => inst_part_num_sel,
                 mnfr_id => inst_mnfr_id,
                 mnfr_id_sel => inst_mnfr_id_sel,
                 clock_dr => clock_dr_inst,
                 shift_dr => shift_dr_inst,
                 update_dr => update_dr_inst,
                 tdo => tdo_inst,
                 tdo_en => tdo_en_inst,
                 tap_state => tap_state_inst,
                 instructions => instructions_inst,
                 sync_capture_en => sync_capture_en_inst,
                 sync_update_dr => sync_update_dr_inst,
                 test => inst_test );


end inst;
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_tap_uc_inst( tck, trst_n, tms, tdi, so,
          bypass_sel, sentinel_val, device_id_sel, user_code_sel, user_code_val,
          ver, ver_sel, part_num, part_num_sel, mnfr_id,
          mnfr_id_sel, clock_dr_inst, shift_dr_inst, update_dr_inst, tdo_inst,
          tdo_en_inst, tap_state_inst, instructions_inst, sync_capture_en_inst,
          sync_update_dr_inst, test );

parameter width = 8;
parameter id = 0;
parameter idcode_opcode = 1;
parameter version = 0;
parameter part = 0;
parameter man_num = 0;
parameter sync_mode = 0;
parameter tst_mode = 1;


input tck;
input trst_n;
input tms;
input tdi;
input so;
input bypass_sel;
input [width-2 : 0] sentinel_val;
input device_id_sel;
input user_code_sel;
input [31 : 0] user_code_val;
input [3 : 0] ver;
input ver_sel;
input [15 : 0] part_num;
input part_num_sel;
input [10 : 0] mnfr_id;
input mnfr_id_sel;
output clock_dr_inst;
output shift_dr_inst;
output update_dr_inst;
output tdo_inst;
output tdo_en_inst;
output [15 : 0] tap_state_inst;
output [width-1 : 0] instructions_inst;
output sync_capture_en_inst;
output sync_update_dr_inst;
input test;

    // Instance of DW_tap_uc
    DW_tap_uc #(width, id, idcode_opcode, version, part, man_num, sync_mode, tst_mode)
      U1 ( .tck(tck), .trst_n(trst_n), .tms(tms), .tdi(tdi), .so(so),
```

```
            .bypass_sel(bypass_sel), .sentinel_val(sentinel_val),
            .device_id_sel(device_id_sel), .user_code_sel(user_code_sel),
            .user_code_val(user_code_val), .ver(ver), .ver_sel(ver_sel),
            .part_num(part_num), .part_num_sel(part_num_sel), .mnfr_id(mnfr_id),
            .mnfr_id_sel(mnfr_id_sel), .clock_dr(clock_dr_inst),
            .shift_dr(shift_dr_inst), .update_dr(update_dr_inst), .tdo(tdo_inst),
            .tdo_en(tdo_en_inst), .tap_state(tap_state_inst),
            .instructions(instructions_inst), .sync_capture_en(sync_capture_en_inst),
            .sync_update_dr(sync_update_dr_inst), .test(test) );

endmodule
```

Revision History

For notes about this release, see the *DesignWare Building Block IP Release Notes*.

For lists of both known and fixed issues for this component, refer to the STAR report.

For a version of this datasheet with visible change bars, click here.

Date	Release	Updates
July 2019	DWBB_201903.3	■ Updated the Verilog example in "HDL Usage Through Component Instantiation - Verilog" on page 11
January 28, 2019	DWBB_201806.5	■ Updated example in "HDL Usage Through Component Instantiation - Verilog" on page 11
January 2019	DWBB_201806.5	■ Updated example in "HDL Usage Through Component Instantiation - VHDL" on page 9 ■ Added this Revision History table and the document links on this page

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.

All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
www.synopsys.com